title: Deploy a Contract description: Deploy your first contract on SUAVE keywords: - contract - deploy - suave - solidity

Deploy A Contract

Now that you have suave-geth running locally, let's deploy a new contract and begin looking at the programming model on SUAVE. Remember, SUAVE enables you to:

- Build blocks on other chains
- Define private compute
- ♀ Store private data
- Access off-chain data
- Interface with many different MEV components

Most of these use cases require that you understand the difference between "onchain" and "offchain" computation and how to use SUAVE's unique features to create applications not otherwise possible on public blockchains. We'll begin with some basic examples which illustrate the difference clearly.

Compatible with Forge

Forge is a smart contract development toolchain we use in our examples. If you do not have it installed, you can get it by running:

bash curl -L https://foundry.paradigm.xyz | bash

Getting started

Begin by creating a new directory and initializing the standard Forge template:

bash mkdir suapp && cd suapp

bash forge init --template https://github.com/foundry-rs/forge-template

You can then open the resulting files in whichever text editor you prefer.

Onchain offchain

Let's adapt the empty contract in the src directory to see the key differences between onchain and offchain computation.

Begin by renaming Contract.sol to MyFirstSuapp.sol and deleting the Contract.t.sol file in the test directory.

Then, you can write two simple functions called onchain() and offchain():

```solidity // SPDX-License-Identifier: Unlicense pragma solidity ^0.8.8;

contract MyFirstSuapp { function onchain() public {}

function offchain() public pure returns (bytes memory) {
    /* This is where you will write all your compute-heavy,
    off-chain logic to be done in a Kettle */
    return abi.encodeWithSelector(this.onchain.selector);
}

} ```

All offchain functions should return a function selector to another function inside your SUAPP, such that the relevant outputs of offchain computation result in onchain state transitions. The Kettle which executes a user's call to offchain() will take the results of any offchain computation and wrap it in a "SUAVE transaction", using the specified function selector. That SUAVE transaction is broadcast onchain, which eventually results in the onchain component being executed too.

Let's take a practical example. In an orderflow auction, users might send their transactions from other domains to a Suapp as confidential inputs. The Suapp can emit specific information about those transactions (without revealing everything) such that searchers listening for events on SUAVE chain can construct backruns. They can then submit their backruns to the same Suapp, which can merge the original transactions and the best backruns offchain (it can take many computational steps do this), before emitting the resulting bundle onchain for a block builder on the original domain to pick up and use.

There are many examples which use this onchain-offchain pattern to achieve results that are not otherwise possible in normal public blockchains. We'll be introducing them one by one through the course of these tutorials.
```

For now, let's just get on with deploying our simple contract.

Deploy Your Contract

Make sure that you still have `suave-geth` running from the previous tutorial. If you're using the latest binary, you can start it with:

```
bash suave-geth --suave.dev
```

If you built from source, you may need to specify the full path instead: `./build/bin/suave-geth --suave.dev`.

Then, compile you new contract with Forge:

```
bash forge build
```

There is a helper facility called `spell` in `suave-geth` which facilitates deploying contracts and sending confidential compute requests. We'll use that to deploy MyFirstSuapp.sol. You can run this command from the top level of your new `suapp` directory:

```
bash suave-geth spell deploy MyFirstSuapp.sol:MyFirstSuapp
```

If you built from source, run:

```
bash ./<path_to_suave-geth>/build/bin/suave-geth spell deploy MyFirstSuapp.sol:MyFirstSuapp
```

You should see a result like this printed to your terminal:

```
bash INFO [03-26|09:59:54.405] Running with local devchain settings INFO [03-26|09:59:54.411] Hash of the result onchain transaction hash=0x487135e51591a911d5bb795ae9969e7e0765386bcc49d988929c447708369f4d INFO [03-26|09:59:54.411] Waiting for the transaction to be mined... INFO [03-26|09:59:54.515] Transaction mined status=1 blockNum=5 INFO [03-26|09:59:54.515] Contract deployed address=0xFcdbc6055c5C36dBc326F308ed74fA8cB00771a6
```

Congratulations! You just deployed your first contract to a local SUAVE network!

Take note of the address to which it has been deployed: you'll need it in the next tutorial, where we'll send confidential compute requests to your contracts, and extend the functions to emit information about offchain computations onchain.